Estimation and Planning Mistakes

- Underestimating complexity, cost, and/or schedule
  - Use historical data and expert judgment to estimate accurately.
- Abandoning planning under pressure
  - Stick to planning to avoid chaotic code-and-fix mode.
- Overly aggressive schedules
  - Set realistic schedules based on historical data and project complexity.
- Wasting time in the \fuzzy front end"
  - Streamline the approval and budgeting process.

Communication and Stakeholder Engagement Mistakes

- Poor communication
  - Hold regular meetings and ensure clear documentation.
- Not engaging stakeholders
  - Include stakeholders in planning and review sessions.
- Insufficient user input
  - Ensure active involvement of end-users throughout the project.

Project Management Mistakes

- Lack of oversight/poor project management
  - Appoint experienced project managers and conduct regular reviews.
- Adding developers to a late project
  - Avoid adding developers late in the project to prevent further delays.

Quality and Risk Management Mistakes

- Poor quality workmanship
  - Implement quality assurance processes and conduct regular code reviews.
- No risk management
  - Identify risks early and develop mitigation plans.
- Ignoring system performance requirements
  - Define and monitor performance requirements throughout the project.
- Poorly planned/managed transitions
  - Develop detailed transition plans and involve all relevant parties.

Recursive vs Incremental vs Iterative Development

- Recursive
  - Repeatedly breaking down a problem into smaller parts until it is simple enough to solve.
  - Example: Divide and conquer.
- Incremental
  - Start by building the core functionality and then add features in subsequent increments.
  - Example: Agile.
- Iterative
- Develop a system through repetition of cycles (iterations)
  - Example: Scrum.

Unified Process

- Workflows
  - Defines activities in process.
  - Each activity has inputs and outputs.
- Phases
  - Inception
  - Elaboration
  - Construction
  - Transition

Cynefin Framework

- Simple
  - Cause and effect are obvious.
  - Best practice.
- Complicated
  - Cause and effect are discoverable.
  - Good practice.
- Complex
  - Cause and effect are only obvious in hindsight.
  - Emergent practice.
- Chaotic
  - No cause and effect relationship.
  - Novel practice.

Predictive vs Adaptive Development

- Predictive
  - Plan-driven.
  - Requirements are stable.
  - Example: Waterfall.
- Adaptive
  - Change-driven.
  - Requirements are volatile.
  - Example: Agile.

Waterfall Model

- Requirements
  - Define system requirements.
- Design
  - Develop system architecture.
- Implementation
  - Write and test code.
- Verification
  - Test system.
- Maintenance
  - Fix bugs and add features.

Agile Manifesto

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Agile Principles

- Satisfy the customer with continuous delivery.
- Welcome changing requirements.
- Frequent delivery of working software.
- Daily collaboration between business and developers.
- Build projects around motivated individuals.
- Face-to-face conversation for communication.
- Working software as progress measure.
- Promote sustainable Dev.
- Continuous attention to technical excellence.
- Simplicity is essential.
- Best architectures emerge from self-organizing teams.
- Regular reflection and adjustment.

Scrum Roles

- Product Owner:
  - Maximizes product value.
  - Develops and communicates Product Goal.
  - Creates and prioritizes Product Backlog.
  - Ensures transparency and understanding of Backlog.
  - One person, not a committee, with leadership role.
- Scrum Master:
  - Facilitates Scrum process, resolves impediments.
  - Creates self-organization environment.
  - Captures empirical data, shields team from distractions.
  - Enforces timeboxes, keeps artifacts visible.
  - Promotes improved practices, has leadership role.
- Dev Team:
  - Develops product, self-organizing, cross-functional.
  - No titles, no sub-teams, no specialized roles.
  - Long-term, full-time membership, 7 ± 2 members.

Scrum Cycle

- Sprint Planning
  - Product Owner presents Product Backlog.
  - Dev Team selects items for Sprint Backlog.
  - Sprint Goal is defined.
- Daily Scrum
  - 15-minute meeting.
  - Dev Team plans work for next 24 hours.
  - Scrum Master enforces timebox.
- Sprint Review
  - Product Owner presents completed work.
  - Dev Team demonstrates work.
  - Stakeholders provide feedback.
- Sprint Retrospective
  - Dev Team reflects on Sprint.
  - Scrum Master facilitates discussion.
  - Team identifies improvements.

Scrum Artifacts

- Product Backlog
  - Prioritized list of features.
  - Updated regularly.
  - Visible to all stakeholders.
  - Owned by Product Owner.
- Sprint Backlog
  - List of tasks for current Sprint.
  - Owned by Dev Team.
  - Updated daily.
  - Created during Sprint Planning Meeting.
  - Decomposed from Product Backlog.
- Burndown Charts
  - Graphical representation of work remaining.
  - Updated daily.
  - Shows progress towards Sprint Goal.
  - Helps identify issues early.
  - Used to forecast project completion.

Increment & Done

- A formal description of the state of the Increment when it meets the quality measures required for the product.
- The moment a Product Backlog item meets the Definition of Done, an Increment is born.
- If a Product Backlog item does not meet the Definition of Done, it cannot be released or even presented at the Sprint Review.

No Silver Bullet

- Scrum will not solve your problems.
- Scrum will make your problems visible.
- You will have to solve your problems.

Accidental vs Essential Complexity

- Essential complexity: - Inherently difficult problems with no known solution.
- Necessary accidental complexity: - Example: project management.
- Unnecessary accidental complexity: - Waste, Lean, MEI (minimum essential information).

Best/Good/Recommended Practices

- "Best Practice": - Consistently improves productivity, cost, schedule, quality, user satisfaction, predictability.
- Best Practices (Glass, 2004): - Dev teams repeat mistakes. - Best practice documents regurgitate textbook material. - Growing field's wisdom not increasing.

Agile Sweet Spots

- Dedicated developers.
- Experienced developers.
- Small co-located team.
- Tools for testing and configuration management.
- Easy user access.
- Short increments and frequent delivery.

Requirements Volatility

- Failure to consider how requirements will change
  - Requirements change about 2% per month for typical project.
  - Change rates of 35-50% for large projects.
  - Typical software project experiences 25% change in requirements.

Requirements Elicitation

- Interviews
  - Structured interviews
    - * Specific preplanned questions are asked.
  - Unstructured interview
    - * Questions are posed in response to the answers received.
- Questions
  - Open-ended questions
    - * Questions are posed to encourage the client to provide more information.
  - Closed-ended questions
    - * Questions are posed to answer specific questions.

Requirements Design

- Functional Requirements
  - Define system behavior.
  - Define what system should do.
- Non-Functional Requirements
  - Describe system properties and constraints.
  - Define how system should do it.

Requirements Analysis

- Functional Decomposition
  - Breaks down system into smaller components.
  - Each component has a specific function.
- Data Flow Diagrams
  - Shows how data flows through system.
  - Identifies sources and destinations of data.
- State Diagrams
  - Shows how system responds to events.
  - Identifies states system can be in.
- Entity-Relationship Diagrams
  - Shows how data is related in system.
  - Identifies entities and relationships.

Characteristics of a good requirement

- Necessary
  - Defines an essential capability, characteristic, constraint, and/or quality factor.
  - If removed, a deficiency will exist.
- Implementation Free
  - Avoids placing unnecessary constraints on the architectural design.
  - States what is required, not how the requirement will be satisfied.
- Unambiguous
  - The requirement is stated in such a way so that it can be interpreted in only one way.
  - The requirement is stated simply and is easy to understand.
- Consistent
  - The requirement is free of conflicts with other requirements.
- Complete
  - The stated requirement needs no further amplification because it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder's need.
- Singular
  - The requirement statement includes only one requirement with no use of conjunctions.
- Feasible
  - The requirement is technically achievable, does not require major technology advances, and fits within system constraints (e.g., cost, schedule, technical, legal, regulatory) with acceptable risk.
- Traceable
  - The requirement is upwards traceable to specific documented stakeholder statement(s) of need, higher tier requirement, or other source (e.g., a trade or design study).
  - The requirement is also downwards traceable to the specific requirements in the lower tier requirements specification or other system definition artefacts.
- Verifiable
  - The requirement has the means to prove that the system satisfies the specified requirement.

Characteristics of a good set of requirements

- Complete
  - Needs no further amplification.
  - Acceptable timeframe for TBD items.
- Consistent
  - No contradictory requirements.
  - No duplicated requirements.
  - Same term used for same item.
- Affordable
  - Can be satisfied by a feasible solution.
  - Within life cycle constraints.
- Bounded
  - Maintains identified scope.
  - Does not increase beyond what is needed.

Use Cases vs User Stories

- Use cases
  - A set of scenarios that identify a thread of usage for the system to be constructed.
  - Tells a stylized story about how an end user interacts with the system under a specific set of circumstances.
  - Captures a contract that describes the system's behavior under various conditions as the system responds to a request from one of its stakeholders.
  - NOT OBJECT ORIENTED.
- User stories
  - A promise to have a discussion; not every detail needs to be included.
  - Describes functionality that will be valuable to either a user or purchaser of a system.
  - Card
    - * Written description of the story used for planning and as a reminder.
  - Conversation
    - * About the story that serve to flesh out the details of the story.
  - Confirmation
    - * Details that can be used to determine when a story is complete.

Use Case Format

- Our user starts by [action]
- Then, the system [response]
- The user then [reacts]
- Finally, the system [result]
- Leaving the user [result]

User Story Format

- As a [role]
- I want [feature]
- So that [benefit]

Facts and Figures

- Requirements volatility
  - 2% per month for typical project.
  - 35-50% for large projects.
  - 25% change in requirements for typical software project.
- Scrum Team Size
  - 7 ± 2 members or < 10 members.